International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 3, Issue 3 (October 2013) PP: 49-54

A Path finding Technique for Open Terrain

Pranav Kumar Pathak[1], Dr. Binod Kumar[2], PhD (CS),
Dr. Dipthi Shah[3], PhD (CS)

ABSTRACT: *Pathfinding involves solving a planning problem with agents seeking optimal paths from a start state to a goal state. The pathfinding process involves utilizing the full state space information available to agents to find the least expensive route to the goal. However most solutions to the pathfinding problem have solely focused on using graph based terrain representations and graph search algorithms to obtain a path. This paper replaces part of the search with a direct geometrical solution. We provide some preliminary indication of the potential merit of the approach.*

Keywords: *AI for games, Steer behavior, Path finding in 3d games and simulation*

I. INTRODUCTION

Pathfinding is a fundamental problem that typical another. It has been defined as the problem of finding a path linking two vertices of a graph [NAH04]. A common solution is to employ the A* search algorithm [HNR68]. The properties of this algorithm are well known. For example, A* is *optimally efficient*, i.e., for a given heuristic, no other optimal algorithm can guarantee expanding fewer nodes than A* [RN03; citing DP85]. Yet, in practice, pathfinding with A* can impose a severe resource impact in terms of memory and time requirements. This is especially true when multiple simultaneous pathfinding requests must be satisfied. The literature abounds with approaches for overcoming the limitations of A*. Common themes overcoming the limitations of A*. Common themes (e.g., grids, waypoints, navigation meshes, etc.), changing the heuristic (e.g., Manhattan distance, Euclidean Distance, Vancouver Distance [Yap02]), pre-computing and/or caching paths, and sacrificing optimality to gain on speed by abstracting the search space (e.g., hierarchical approaches [BMS04]).

Many approaches to the pathfinding problem model the underlying world as a grid. The squares in the grid are then represented as the nodes of a search graph. Performance improvements are then sought using graph-theoretic algorithms and data structures. It is important to keep in mind that the graph representation is not a faithful model of the world; it fails to capture important spatial information (e.g., spatial proximity is not explicitly represented---only connectedness). An optimal solution in the graph is only an approximation of the optimal solution in the connectedness (typically 4 or 8 directions).

The lack of geo-awareness of such representations becomes most obvious in open terrain environments. In contrast to maze-like environments where A*'s expanding search fringe is constrained by walls and obstructions, in open terrains, expansion is only constrained by the heuristic and terrain cost. To find a path of a given length in the open is typically much more expensive than finding a path of the same length in a constrained environment. Approaches which are feasible for the confines of a dungeon fall down in the light of day. This paper presents a preliminary proposal for an approach to pathfinding in open terrains that seeks to improve existing approaches by exploiting geometric information derived from the world model. While the approach is aimed at open terrain environments, it is equally applicable (without modification) to the deepest, darkest dungeons as well.

II. PATHFINDING IN PRACTICE

A* search is among the most popular pathfinding techniques adopted by commercial game developers. In conjunction with A* search, many different search space representations (each having advantages and disadvantages) have been tried. A useful introduction to typical search space representations is given by Tozour [Toz04].

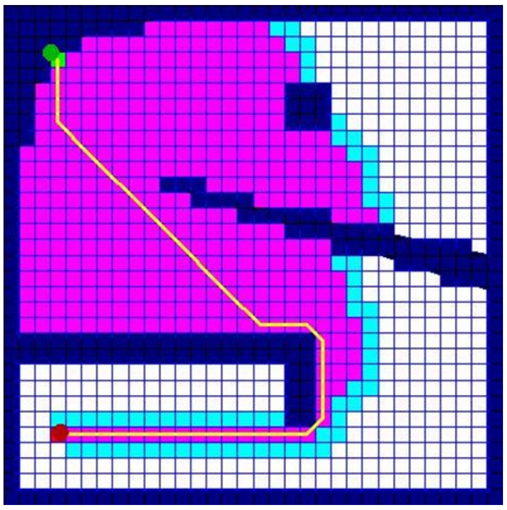
Figure 1: Grid representation [Toz05]

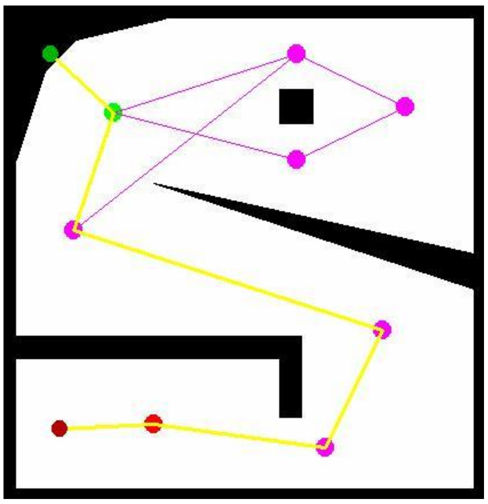
Figure 2: Waypoint representation [Toz05]

As Tozour explains, the simplest representation of the game world is a regular grid of squares, hexes, or triangles. Grids are most useful in 2D worlds. They support random-access lookup, that is, given the coordinates of a point in the world, one can directly determine the cell that contains it. However, since the size of the grid's cells impacts the quality of the solution path, grids typically require a large number of cells to adequately represent the underlying world (hence, a large search space). Techniques such as 'string-pulling' [Sno00] or spline fitting [Rab00] can be used to provide more natural looking paths.

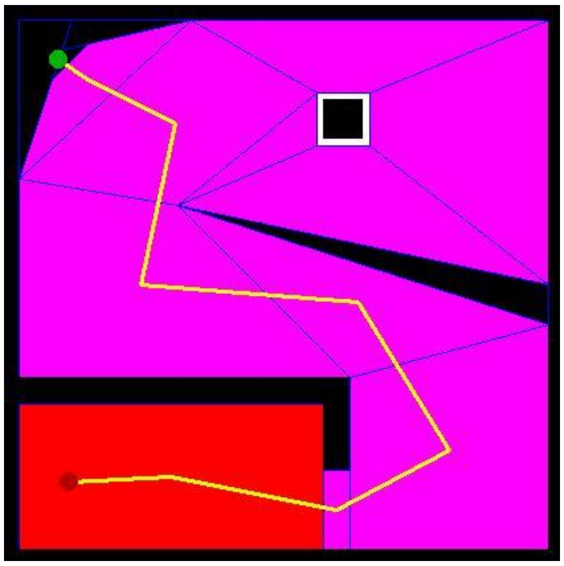
Figure 3: Navigation mesh representation [Toz05]

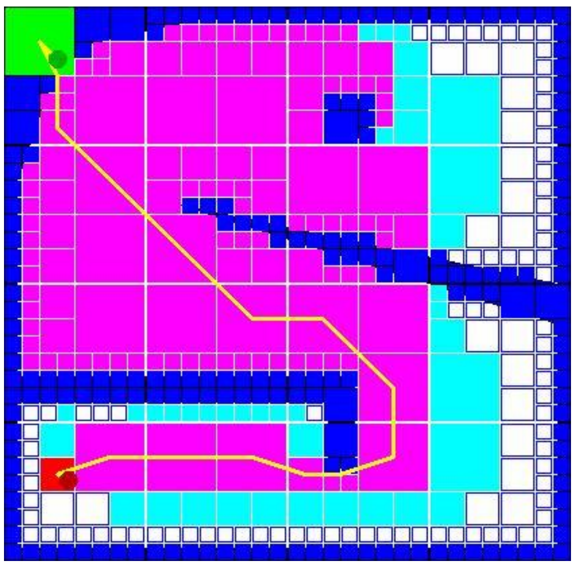
Figure 4: Quad tree representation [Toz05]

The advantages and disadvantages of representations such as waypoint graphs and navigation meshes are also covered by Tozour. The key issue is the trade- off between the size of the search space and the quality of the path found.

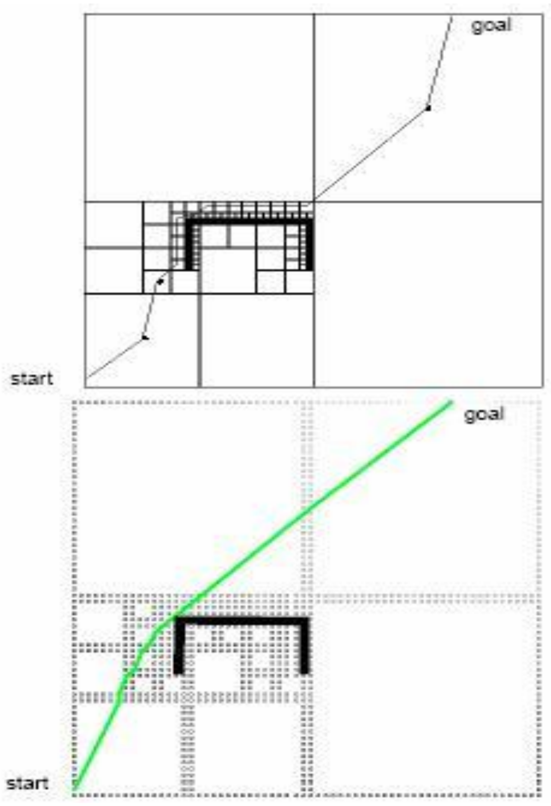

Figure 5: Framed quad trees [YSSB98].

In unstructured open terrains, this trade-off is acute. One approach to this problem is the use of a quad-tree representation. The map is subdivided into 4 squares. Each square that contains an obstacle is subdivided again. The process is repeated recursively until the area of a square reaches a specified minimum. The advantage of this approach is that large unobstructed areas are represented as a single node. The disadvantage is that the quality of the path is reduced.

Compare the nodes expanded versus path quality in Figures 1 to 4. The start and goal points are green and red, respectively.

The start and end graph nodes are also green and red. The explored nodes (closed list) are indicated in purple and the fringe nodes (open list) are in cyan.

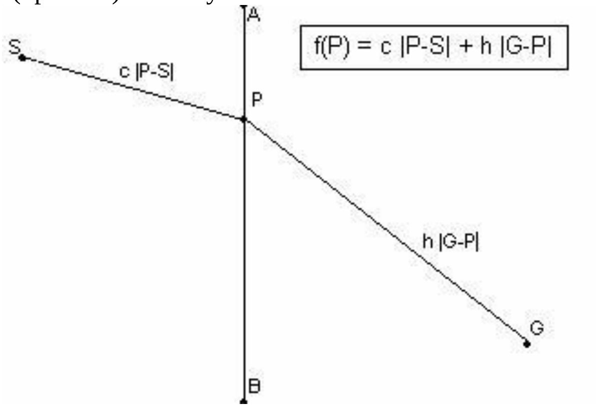

Figure 6: Find optimal boundary point problem.

Note that in the quad tree approach, the path is constructed from quad center to quad center. For large quads, this can dramatically affect the path quality. To overcome this limitation yet attempt to retain, to a degree, the reduced search space advantage, another approach called framed quad trees has been proposed [CSU97]. In this approach, high resolution cells adjoin the boundaries of the quads. Paths passing through a quad travel from boundary to boundary, crossing at a high resolution cell rather than going center to center. Compare the path quality of framed quad trees versus quad trees in the illustration from Yahja et al. [YSSB98] in Figure 5. The improved path quality comes at the price of larger search space for cluttered maps.

One further approach is to partition the terrain into convex polygons covering regions of uniform terrain. Stout discusses several ways of doing this [Sto00]. An advantage of this representation is that because the polygons are convex and uniform cost, the locally minimal path traversing the covered region is a straight line. Since this line is guaranteed to remain in the confines of the polygon, its cost is simply its Euclidean length times the region cost.

In all the representations discussed here, the ultimate result is a graph to be searched using the A* algorithm which relies on a given cost function and heuristic cost estimator. One possibility for improvement is to provide a means to take advantage of geometric properties of the underlying world
directly, rather than indirectly through search.

III. OUR APPROACH

As a starting point, consider a representation that is a hybrid of framed quad trees and the convex polygon approach. Instead of treating the regions as nodes in the search graph, let the polygon edges be divided nto a number of points based on the desired resolution. Take each of these points as nodes in the search graph. Nodes are connected if they correspond to points in the same polygon (points on the boundary between two polygons are each represented by a single node that belong to both lygons). The cost of the edges is now simply the Euclidean distance between them times the cost of the region covered by their polygon. Clearly, the properties of search using this representation are similar to that of framed quad trees but with some additional flexibility from using convex polygons rather than quads. Such an approach would be useful for open terrain environments except that, as the resolution increases, the search cost increases dramatically. To seek an improvement to this, consider the fact that the optimal path in a given search graph passes through the boundaries of a sequence of polygons. The path crosses each boundary at a point (or possibly through multiple points on a boundary). To determine which boundary points to cross the f-score for each corresponding node is calculated and nodes are selected in the usual A* fashion.

In Figure 6, imagine S is the starting point which lies in a region with a boundary AB which is divided into points $\{A,P_1,...,P,\dots, P_n,B\}$. Let G be the goal point in some possibly remote region. Let c be the terrain cost in the region containing S and let h be the heuristic cost factor (typically, h=1). For each, point on the boundary (and points on the other boundaries of the region containing S), A* must calculate the f- score of the point in order to later choose the next node to expand. For a point P on AB,
$f(P) = g(P)+h(P) = c|P-S|+h|G-P|$.

Rather than determine the f-score of each point and thereby indirectly determine the point P on AB that minimizes the f-score, it is possible to determine P directly by an application of Snell's Law of Refraction of Light (we independently rediscovered this idea originally put forth by Mitchell and Papadimitriou [MP91]). According to Snell's Law, the path of a light ray passing through a boundary AB between regions with indices of refraction c and h satisfies the equation $c\cdot\sin\theta = h\cdot\sin\varphi$ where θ and φ are the angles of incidence and refraction respectively. Snell's Law is derived from Fermat's Principle of Least Time (light follows the path of least time). In [MP91], an algorithm using a continuous Dijkstra technique simulates the effect of a wave front emanating from S.

Our approach differs in that we integrate a boundary point solver (based on Snell's Law) into the A* algorithm. The idea is to dramatically reduce the search space by eliminating the need to evaluate individual boundary points while retaining (and even improving) path quality. To do this, we need to solve the problem of determining the boundary points on each of the boundaries a path cross between S and G by minimizing a heuristic cost function of the form $(P_1,\dots,P_k) = \sum c_{k-1}|P_k-P_{k-1}| + h|G-P_k|$ where P_0=S. Figure7 illustrates this for k=2 (the blue path).

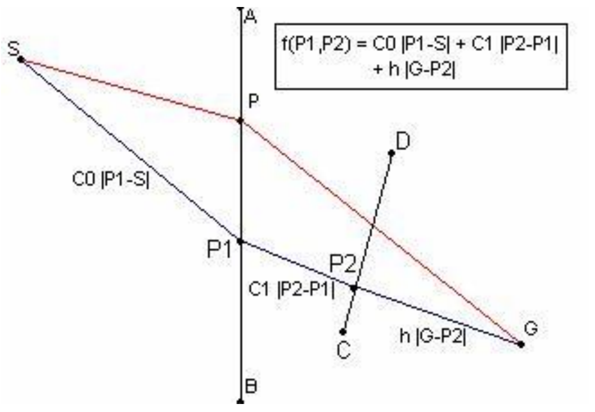

FIGURE 7: FINDING OPTIMAL BOUNDARY POINTS

IV. BPSOLVER

In mathematical terms, minimizing the multivariable cost function produces in a system consisting of k quartic equations in k unknowns. Mitchell and Papadimitriou [MP91] have noted that elimination among these equations yields a polynomial of degree doubly exponential in k. Instead of seeking an exact solution, we adopt a commonly used optimization method known as iterative coordinate decent which works by iteratively updating the points to minimize the cost function.

At the heart of this iterative procedure (which we call BPSolver), is a function, BPCompute One, that computes the exact solution for the locally optimal boundary point P given the prior and next point and prior and next region costs. In Figure 8, given P1 and G, BPCompute One finds P_2 on CD which minimizes $f(P_1,P_2)$. It does this by finding the roots of the quartic equation that results from differentiating f with respect to P_2 (expressed in parametric form $P_2(t) = C + t(D-C)$ for $0 \leq t \leq 1$) and setting the derivative equal to zero. Using Eberly's root solver [Ebe05], the cost function is evaluated at roots of this equation (or the closest endpoint if the root is out of range) to choose the minimum. The quartic equation is of the form $w_4rt^4+w_3t^3+w_2t^2+w_1t+w_0=0$ where the w_i are constants whose values are obtained by plugging in the values of P_1, G, c_1, h, C, and D into simple (but tedious to write out) formulas obtained by straightforward application of calculus and algebra (see Figure 8).

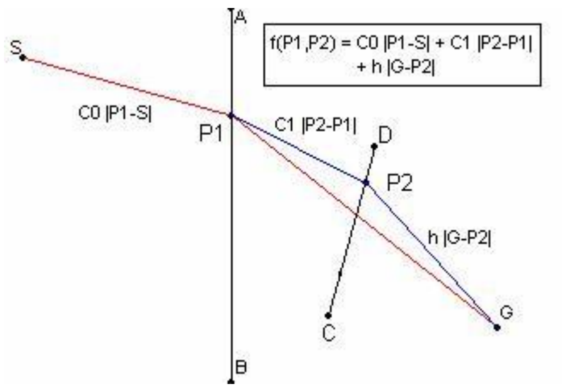

FIGURE 8: BP COMPUTE ONE FUNCTION

BPSolver makes use of the local minimum found by BPCompute to obtain the globally minimal solution by iterative coordinate descent. We prime our procedure with the previously obtained solution for k-1 points. We introduce the k-th point. The initial value for this point is determined by a call to BPCompute. After each call to BPCompute, the points on the prior and next boundaries (not including the start and goal points which are fixed) are added to a Change Set for re-evaluation in the next iteration. Points in the Change Set are revised by calls to BPCompute. The iterations continue until the improvement of overall cost function is less than some given threshold (or a maximum number of iterations are reached or the Change Set is empty). Mitchell and Papadimitriou [MP91] have noted that due to the convexity of the cost function, iterative coordinate descent procedures such as this will converge to the global optimum, but the bound on the number of iterations required to achieve a solution within epsilon of optimal are not known. They cite some empirical results that suggest that convergence will typically be fast in practice.

V. USING BP SOLVER IN A

Armed with BPSolver, our next task is to integrate it into the A* search algorithm. Recall the boundary points as nodes search graph representation described at the start of Section 3. To avoid the combinatorial growth in the search tree that results from the high branching factor in this representation, we instead use boundaries as nodes in the search graph. The twist we add is that boundary nodes keep track of the optimum boundary point so far. Of course, this point will change as the A* search progresses and the cost will change as the A* search progresses and the cost in the cost function The A* search begins in the usual way by placing the start node on the OPEN list. We treat the start point as a point on a boundary of length 0. Boundaries are treated as two sided and directed (connecting a from- and a to-region). The start node is dynamically inserted into the search graph with its to-region being the polygon that contains the start point. The goal node is treated similarly. The cost function for the start node is computed (simply 0+h|G-S|). When a node is selected from the OPEN list for expansion its successors are boundaries of the node's to-region. A node is related for each such boundary. At this point, BPSolver is called to determine the optimal point on the node's boundary. As a consequence, we have also determined the node's f-score. These newly evaluated nodes are inserted into the OPEN list. From here, the A* search proceeds in the usual Fashion.

There is a complication that must be handled. In typical implementations of A*, the f-score for a child determined by summing the g-score of the parent and the h-score of the child. This will not work here because the g-score of the parent may be unrelated to the path determined by BPSolver for the child. This is because the boundary points used in calculating the g-score of the parent may be revised by BPSolver in the child when taking into account the actual cost of the next region that replaces the previous heuristic estimate. In fact, the parent of a node is in the search tree is not really the node's parent in the usual sense.

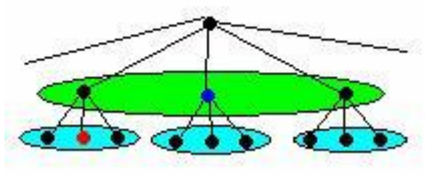

FIGURE 9: BOUNDARY POINTS AS NODES

To see this, compare the boundary points as nodes representation (BPrep) in Figure 9 versus boundaries as nodes representation (Brep) in Figure 10.

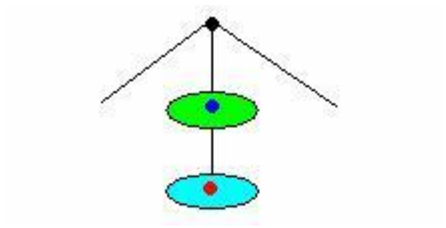

FIGURE 10: BOUNDARIES AS NODES.

The figures depict part of the search tree. The green ellipse encloses nodes corresponding to a boundary. The cyan ellipse(s) enclose(s) nodes corresponding to a neighbouring boundary. In Figure 9, because points are nodes, each boundary has multiple nodes

REFERENCES

[1] Barwood, H. & Falstein, N. 2002. .More of the 400: Discovering Design Rules.. Lecture at *Game Developers Conference*, 2002. Available online at: http://www.gdconf.com/archives/2002/hal_barwood.ppt

[2] Church, D. 1999. .Formal Abstract Design Tools.. *Game Developer*, August 1999. San Francisco, CA: CMP Media. Available online at: http://www.gamasutra.com/features/19990716/design_tools_01.htm

[3] Hunicke, R. 2004. .AI Babysitter Elective.. Lecture at *Game Developers Conference Game Tuning Workshop*, 2004. In LeBlanc et al., 2004a. Available online at: http://algorithmancy.8kindsoffun.com/GDC2004/AITutorial5.ppt LeBlanc, M., ed. 2004a. .Game Design and Tuning

[4] Workshop Materials., *Game Developers Conference 2004*.Available online at: http://algorithmancy.8kindsoffun.com/GDC2004/

[5] LeBlanc, M. 2004b. .Mechanics, Dynamics, Aesthetics: A Formal Approach to Game Design.. Lecture at Northwestern University, April 2004.

About Authors

Pranav Kumar Pathak

I have done Masters from Sardar Patel University,India. Currently pursuing Ph.D in Computer Science (AI) from Sardar Patel University, India. I have also done MBA IT from Manipal University, India. Currently I am working as Lecturer in Computer Science Department at College of Computer and Information Sciences in King Saud University, Kingdome of Saudi Arabia. I have presented several papers at international Journals and conferences.

Dr. Binod Kumar

I have done My Ph.D in Computer Science. I have also done M. Phil (CS), MiEEE, MIAENG. Currently I am working as Dean-MCA and Professor in JSPM's Jayawant Technical Campus, Pune, INDIA www.jspm.edu.in

Dr. Dipti M. Shah

I have done Ph.D in Computer Science from Sardar Patel University, India. Currently I am working as Associate Professor in Computer Science Department at G.H. Patel Pg Department Of Computer Science & Technology V. V. Nagar. I have presented several papers at international Journals and conferences.